[Chapter 1From zero to deploy](http://ruby.railstutorial.org/chapters/beginning" \l "top)

Welcome to the [*Ruby on Rails Tutorial*](http://ruby.railstutorial.org/ruby-on-rails-tutorial-book). The goal of this book is to be the best answer to the question, “If I want to learn web development with [Ruby on Rails](http://rubyonrails.org/), where should I start?” By the time you finish the *Ruby on Rails Tutorial*, you will have all the skills you need to develop and deploy your own custom web applications with Rails. You will also be ready to benefit from the many more advanced books, blogs, and screencasts that are part of the thriving Rails educational ecosystem. Finally, since the *Ruby on Rails Tutorial* uses Rails 4, the knowledge you gain here represents the state of the art in web development. (The most up-to-date version of the *Ruby on Rails Tutorial* can be found on the book’s website at <http://railstutorial.org/>; if you are reading this book offline, be sure to check the [online version of the Rails Tutorial book](http://railstutorial.org/book) at<http://railstutorial.org/book> for the latest updates.)

(*Note*: The present volume is the Rails 4.0 *version* of the book, which means that it has been revised to be compatible with Rails 4.0, but it is not yet a new *edition* because the changes in Rails don’t yet justify it. From the perspective of an introductory tutorial, the differences between Rails 4.0 and the previous version, Rails 3.2, are slight. Indeed, although there are a large number of miscellaneous small changes ([Box 1.1](http://ruby.railstutorial.org/chapters/beginning#sidebar-diffs)), for our purposes there is only one significant difference, a new security technique called *strong parameters*, covered in [Section 7.3.2](http://ruby.railstutorial.org/chapters/sign-up#sec-strong_parameters). Once the changes in Rails justify the effort, I plan to prepare a full new edition of the *Rails Tutorial*, including coverage of topics such as *Turbolinks* and *Russian doll caching*, as well as some new aspects of RSpec, such as *feature specs*.)

**Box 1.1.Diffs from the 2nd edition**

This is a (nearly) comprehensive list of differences between the 2nd edition of the *Ruby on Rails Tutorial* and the present version. (The only really important one is the change to strong parameters; the others are all relatively minor.) This list is presented for the convenience of those who read the 2nd edition (or are otherwise familiar with Rails 3.2) and want a summary of the diffs. If you don’t already have experience with Rails 3.2, you should probably ignore this list.

In what follows, each item includes a reference to a section or code listing with an example of the change in question.

* Change Rails 3.2 to Rails 4.0 ([Section 1.2.2](http://ruby.railstutorial.org/chapters/beginning#sec-rubygems))
* Explicitly include Capybara DSL ([Listing 3.10](http://ruby.railstutorial.org/chapters/static-pages#code-capybara_dsl))
* Change RSpec **.should** to **expect().to** ([Section 3.2.1](http://ruby.railstutorial.org/chapters/static-pages#sec-TDD))
* Change **have\_selector(’title’, …)** to **have\_title(…)** ([Section 3.3.1](http://ruby.railstutorial.org/chapters/static-pages#sec-testing_a_title_change))
* Change HTTP verb from PUT to PATCH for updates ([Box 3.3](http://ruby.railstutorial.org/chapters/static-pages#sidebar-get_etc))
* Add hash arguments for Turbolinks to stylesheets and JavaScripts ([Listing 3.26](http://ruby.railstutorial.org/chapters/static-pages#code-application_layout))
* Change **root to: ’path’** to **root ’path’** ([Listing 5.26](http://ruby.railstutorial.org/chapters/filling-in-the-layout#code-root_route))
* Change **find\_by\_thing(…)** to **find\_by(thing: …)** ([Section 6.1.4](http://ruby.railstutorial.org/chapters/modeling-users#sec-finding_user_objects))
* Switch from **rake db:test:prepare** to **rake test:prepare** ([Section 6.2.1](http://ruby.railstutorial.org/chapters/modeling-users#sec-initial_user_tests))
* Change from **attr\_accessible** to strong parameters ([Section 7.3.2](http://ruby.railstutorial.org/chapters/sign-up#sec-strong_parameters))
* Change to encrypted remember tokens ([Section 8.2.1](http://ruby.railstutorial.org/chapters/sign-in-sign-out#sec-remember_me))
* Change **before\_filter** to **before\_action** ([Listing 9.12](http://ruby.railstutorial.org/chapters/updating-showing-and-deleting-users#code-authorize_before_filter))
* Use Capybara’s **match: :first** to click on the first matching link ([Listing 9.42](http://ruby.railstutorial.org/chapters/updating-showing-and-deleting-users#code-delete_link_tests))
* Change **default\_scope** from a hash argument to a lambda ([Listing 10.11](http://ruby.railstutorial.org/chapters/user-microposts#code-micropost_ordering))
* Change **dup** to **to\_a** ([Listing 10.12](http://ruby.railstutorial.org/chapters/user-microposts#code-micropost_dependency_test))
* Use XPath to test button toggling ([Section 11.2.4](http://ruby.railstutorial.org/chapters/following-users#sec-a_working_follow_button_the_standard_way))

It’s worth emphasizing that the goal of this book is *not* merely to teach Rails, but rather to teach*web development with Rails*, which means acquiring (or expanding) the skills needed to develop software for the World Wide Web. In addition to Ruby on Rails, this skillset includes HTML & CSS, databases, version control, testing, and deployment. To accomplish this goal, the *Ruby on Rails Tutorial* takes an integrated approach: you will learn Rails by example by building a substantial sample application from scratch. As [Derek Sivers](http://sivers.org/) notes in the foreword, this book is structured as a linear narrative, designed to be read from start to finish. If you are used to skipping around in technical books, taking this linear approach might require some adjustment, but I suggest giving it a try. You can think of the *Ruby on Rails Tutorial* as a video game where you are the main character, and where you level up as a Rails developer in each chapter. (The exercises are the [minibosses](http://en.wikipedia.org/wiki/Boss_(video_gaming)" \l "Miniboss).)

In this first chapter, we’ll get started with Ruby on Rails by installing all the necessary software and by setting up our development environment ([Section 1.2](http://ruby.railstutorial.org/chapters/beginning#sec-up_and_running)). We’ll then create our first Rails application, called (appropriately enough) **first\_app**. The *Rails Tutorial* emphasizes good software development practices, so immediately after creating our fresh new Rails project we’ll put it under version control with Git ([Section 1.3](http://ruby.railstutorial.org/chapters/beginning#sec-version_control)). And, believe it or not, in this chapter we’ll even put our first app on the wider web by *deploying* it to production ([Section 1.4](http://ruby.railstutorial.org/chapters/beginning#sec-deploying)).

In [Chapter 2](http://ruby.railstutorial.org/chapters/a-demo-app#top), we’ll make a second project, whose purpose is to demonstrate the basic workings of a Rails application. To get up and running quickly, we’ll build this *demo app* (called **demo\_app**) using scaffolding ([Box 1.2](http://ruby.railstutorial.org/chapters/beginning#sidebar-scaffolding)) to generate code; since this code is both ugly and complex, [Chapter 2](http://ruby.railstutorial.org/chapters/a-demo-app#top) will focus on interacting with the demo app through its *URIs* (often called *URLs*)[1](http://ruby.railstutorial.org/chapters/beginning#fn-1_1) using a web browser.

The rest of the tutorial focuses on developing a single large *sample application* (called**sample\_app**), writing all the code from scratch. We’ll develop the sample app using *test-driven development* (TDD), getting started in [Chapter 3](http://ruby.railstutorial.org/chapters/static-pages#top) by creating static pages and then adding a little dynamic content. We’ll take a quick detour in [Chapter 4](http://ruby.railstutorial.org/chapters/rails-flavored-ruby#top) to learn a little about the Ruby language underlying Rails. Then, in [Chapter 5](http://ruby.railstutorial.org/chapters/filling-in-the-layout#top) through [Chapter 9](http://ruby.railstutorial.org/chapters/updating-showing-and-deleting-users#top), we’ll complete the foundation for the sample application by making a site layout, a user data model, and a full registration and authentication system. Finally, in [Chapter 10](http://ruby.railstutorial.org/chapters/user-microposts#top) and [Chapter 11](http://ruby.railstutorial.org/chapters/following-users#top) we’ll add microblogging and social features to make a working example site.

The final sample application will bear more than a passing resemblance to a certain popular [social microblogging site](http://twitter.com/)—a site which, coincidentally, was also originally written in Rails. Though of necessity our efforts will focus on this specific sample application, the emphasis throughout the *Rails Tutorial* will be on general principles, so that you will have a solid foundation no matter what kinds of web applications you want to build.

**Box 1.2.Scaffolding: Quicker, easier, more seductive**

From the beginning, Rails has benefited from a palpable sense of excitement, starting with the famous [15-minute weblog video](http://www.youtube.com/watch?v=Gzj723LkRJY) by Rails creator David Heinemeier Hansson. That video and its successors are a great way to get a taste of Rails’ power, and I recommend watching them. But be warned: they accomplish their amazing fifteen-minute feat using a feature called*scaffolding*, which relies heavily on *generated code*, magically created by the Rails **generate**command.

When writing a Ruby on Rails tutorial, it is tempting to rely on the scaffolding approach—it’s[quicker, easier, more seductive](http://en.wikipedia.org/wiki/Dark_side_(Star_Wars)). But the complexity and sheer amount of code in the scaffolding can be utterly overwhelming to a beginning Rails developer; you may be able to use it, but you probably won’t understand it. Following the scaffolding approach risks turning you into a virtuoso script generator with little (and brittle) actual knowledge of Rails.

In the *Ruby on Rails Tutorial*, we’ll take the (nearly) polar opposite approach: although[Chapter 2](http://ruby.railstutorial.org/chapters/a-demo-app#top) will develop a small demo app using scaffolding, the core of the *Rails Tutorial* is the sample app, which we’ll start writing in [Chapter 3](http://ruby.railstutorial.org/chapters/static-pages#top). At each stage of developing the sample application, we will write *small, bite-sized* pieces of code—simple enough to understand, yet novel enough to be challenging. The cumulative effect will be a deeper, more flexible knowledge of Rails, giving you a good background for writing nearly any type of web application.

[1.1 Introduction](http://ruby.railstutorial.org/chapters/beginning#sec-introduction)

Since its debut in 2004, Ruby on Rails has rapidly become one of the most powerful and popular frameworks for building dynamic web applications. Everyone from scrappy startups to huge companies have used Rails: [37signals](http://37signals.com/), [GitHub](http://github.com/), [Shopify](http://shopify.com/), [Scribd](http://scribd.com/), [Twitter](http://twitter.com/), [Disney](http://disney.com/), [Hulu](http://hulu.com/), the [Yellow Pages](http://yellowpages.com/)—the [list of sites using Rails](http://rubyonrails.org/applications) goes on and on. There are also many web development shops that specialize in Rails, such as [ENTP](http://entp.com/), [thoughtbot](http://thoughtbot.com/), [Pivotal Labs](http://pivotallabs.com/), and [Hashrocket](http://hashrocket.com/), plus innumerable independent consultants, trainers, and contractors.

What makes Rails so great? First of all, Ruby on Rails is 100% open-source, available under the permissive [MIT License](http://www.opensource.org/licenses/mit-license.php), and as a result it also costs nothing to download or use. Rails also owes much of its success to its elegant and compact design; by exploiting the malleability of the underlying [Ruby](http://ruby-lang.org/) language, Rails effectively creates a [domain-specific language](http://en.wikipedia.org/wiki/Domain_Specific_Language) for writing web applications. As a result, many common web programming tasks—such as generating HTML, making data models, and routing URLs—are easy with Rails, and the resulting application code is concise and readable.

Rails also adapts rapidly to new developments in web technology and framework design. For example, Rails was one of the first frameworks to fully digest and implement the REST architectural style for structuring web applications (which we’ll be learning about throughout this tutorial). And when other frameworks develop successful new techniques, Rails creator [David Heinemeier Hansson](http://loudthinking.com/) and the [Rails core team](http://rubyonrails.org/core) don’t hesitate to incorporate their ideas. Perhaps the most dramatic example is the merger of Rails and Merb, a rival Ruby web framework, so that Rails now benefits from Merb’s modular design, stable [API](http://en.wikipedia.org/wiki/Application_programming_interface), and improved performance.

Finally, Rails benefits from an unusually enthusiastic and diverse community. The results include hundreds of open-source [contributors](http://contributors.rubyonrails.org/), well-attended [conferences](http://railsconf.com/), a huge number of [gems](https://rubygems.org/) (self-contained solutions to specific problems such as pagination and image upload), a rich variety of informative blogs, and a cornucopia of discussion forums and IRC channels. The large number of Rails programmers also makes it easier to handle the inevitable application errors: the “Google the error message” algorithm nearly always produces a relevant blog post or discussion-forum thread.

[1.1.1 Comments for various readers](http://ruby.railstutorial.org/chapters/beginning#sec-comments_for_various_readers)

The *Rails Tutorial* contains integrated tutorials not only for Rails, but also for the underlying Ruby language, the RSpec testing framework, [HTML](http://en.wikipedia.org/wiki/HTML), [CSS](http://en.wikipedia.org/wiki/CSS), a small amount of [JavaScript](http://en.wikipedia.org/wiki/JavaScript), and even a little[SQL](http://en.wikipedia.org/wiki/SQL). This means that, no matter where you currently are in your knowledge of web development, by the time you finish this tutorial you will be ready for more advanced Rails resources, as well as for the more systematic treatments of the other subjects mentioned. It also means that there’s a *lot*of material to cover; if you don’t already have much experience programming computers, you might find it overwhelming. The comments below contain some suggestions for approaching the *Rails Tutorial* depending on your background.

**All readers:** One common question when learning Rails is whether to learn Ruby first. The answer depends on your personal learning style and how much programming experience you already have. If you prefer to learn everything systematically from the ground up, or if you have never programmed before, then learning Ruby first might work well for you, and in this case I recommend[*Beginning Ruby*](http://www.amazon.com/gp/product/1430223634) by Peter Cooper. On the other hand, many beginning Rails developers are excited about making *web* applications, and would rather not slog through a 500-page book on pure Ruby before ever writing a single web page. In this case, I recommend following the short interactive tutorial at [Try Ruby](http://tryruby.org/),[2](http://ruby.railstutorial.org/chapters/beginning" \l "fn-1_2) and then optionally do the free tutorial at [Rails for Zombies](http://railsforzombies.org/)[3](http://ruby.railstutorial.org/chapters/beginning#fn-1_3) to get a taste of what Rails can do.

Another common question is whether to use tests from the start. As noted in the introduction, the*Rails Tutorial* uses test-driven development (also called test-first development), which in my view is the best way to develop Rails applications, but it does introduce a substantial amount of overhead and complexity. If you find yourself getting bogged down by the tests, I suggest either skipping them on a first reading or (even better) using them as a tool to verify your code’s correctness without worrying about how they work. This latter strategy involves creating the necessary test files (called *specs*) and filling them with the test code *exactly* as it appears in the book. You can then run the test suite (as described in [Chapter 5](http://ruby.railstutorial.org/chapters/filling-in-the-layout#top)) to watch it fail, then write the application code as described in the tutorial, and finally re-run the test suite to watch it pass.

**Inexperienced programmers:** The *Rails Tutorial* is not aimed principally at beginning programmers, and web applications, even relatively simple ones, are by their nature fairly complex. If you are completely new to web programming and find the *Rails Tutorial* too difficult, I suggest learning the basics of HTML and CSS and then giving the *Rails Tutorial* another go. (Unfortunately, I don’t have a personal recommendation here, but [*Head First HTML*](http://headfirstlabs.com/books/hfhtml/) looks promising, and one reader recommends [*CSS: The Missing Manual*](http://www.amazon.com/gp/product/0596526873) by David Sawyer McFarland.) You might also consider reading the first few chapters of [*Beginning Ruby*](http://www.amazon.com/gp/product/1430223634) by Peter Cooper, which starts with sample applications much smaller than a full-blown web app. That said, a surprising number of beginners have used this tutorial to learn web development, so I suggest giving it a try, and I especially recommend the [*Rails Tutorial* screencast series](http://railstutorial.org/screencasts)[4](http://ruby.railstutorial.org/chapters/beginning#fn-1_4) to give you an “over-the-shoulder” look at Rails software development.

**Experienced programmers new to web development:** Your previous experience means you probably already understand ideas like classes, methods, data structures, etc., which is a big advantage. Be warned that if your background is in C/C++ or Java, you may find Ruby a bit of an odd duck, and it might take time to get used to it; just stick with it and eventually you’ll be fine. (Ruby even lets you put semicolons at the ends of lines if you miss them too much.) The *Rails Tutorial* covers all the web-specific ideas you’ll need, so don’t worry if you don’t currently know aPOST from a PATCH.

**Experienced web developers new to Rails:** You have a great head start, especially if you have used a dynamic language such as PHP or (even better) Python. The basics of what we cover will likely be familiar, but test-driven development may be new to you, as may be the structured REST style favored by Rails. Ruby has its own idiosyncrasies, so those will likely be new, too.

**Experienced Ruby programmers:** The set of Ruby programmers who don’t know Rails is a small one nowadays, but if you are a member of this elite group you can fly through this book and then move on to developing applications of your own.

**Inexperienced Rails programmers:** You’ve perhaps read some other tutorials and made a few small Rails apps yourself. Based on reader feedback, I’m confident that you can still get a lot out of this book. Among other things, the techniques here may be more up-to-date than the ones you picked up when you originally learned Rails.

**Experienced Rails programmers:** This book is unnecessary for you, but many experienced Rails developers have expressed surprise at how much they learned from this book, and you might enjoy seeing Rails from a different perspective.

After finishing the *Ruby on Rails Tutorial*, I recommend that experienced programmers read [*The Well-Grounded Rubyist*](http://www.amazon.com/gp/product/1933988657) by David A. Black, [*Eloquent Ruby*](http://www.amazon.com/Eloquent-Ruby-Addison-Wesley-Professional-Series/dp/0321584104/) by Russ Olsen, or [*The Ruby Way*](http://www.amazon.com/gp/product/0672328844) by Hal Fulton, which is also fairly advanced but takes a more topical approach.

At the end of this process, no matter where you started, you should be ready for the many more intermediate-to-advanced Rails resources out there. Here are some I particularly recommend:

* [RailsCasts](http://railscasts.com/) by Ryan Bates: Excellent (mostly) free Rails screencasts
* [PeepCode](http://peepcode.com/): Excellent commercial screencasts
* [Code School](http://www.codeschool.com/): Interactive programming courses
* [Rails Guides](http://guides.rubyonrails.org/): Good topical and up-to-date Rails references
* [RailsCasts](http://railscasts.com/) by Ryan Bates: Did I already mention [RailsCasts](http://railscasts.com/)? Seriously: *[RailsCasts](http://railscasts.com/)*.

[1.1.2 “Scaling” Rails](http://ruby.railstutorial.org/chapters/beginning#sec-1_1_2)

Before moving on with the rest of the introduction, I’d like to take a moment to address the one issue that dogged the Rails framework the most in its early days: the supposed inability of Rails to “scale”—i.e., to handle large amounts of traffic. Part of this issue relied on a misconception; [you scale a *site*, not a framework](http://idleprocess.wordpress.com/2009/11/24/presentation-summary-high-performance-at-massive-scale-lessons-learned-at-facebook/), and Rails, as awesome as it is, is only a framework. So the real question should have been, “Can a site built with Rails scale?” In any case, the question has now been definitively answered in the affirmative: some of the most heavily trafficked sites in the world use Rails. Actually *doing* the scaling is beyond the scope of just Rails, but rest assured that if *your*application ever needs to handle the load of Hulu or the Yellow Pages, Rails won’t stop you from taking over the world.

[1.1.3 Conventions in this book](http://ruby.railstutorial.org/chapters/beginning#sec-conventions)

The conventions in this book are mostly self-explanatory. In this section, I’ll mention some that may not be.

Both the [HTML](http://railstutorial.org/book) and [PDF](http://railstutorial.org/) editions of this book are full of links, both to internal sections (such as[Section 1.2](http://ruby.railstutorial.org/chapters/beginning#sec-up_and_running)) and to external sites (such as the main [Ruby on Rails download](http://rubyonrails.org/download) page).[5](http://ruby.railstutorial.org/chapters/beginning#fn-1_5)

Many examples in this book use command-line commands. For simplicity, all command line examples use a Unix-style command line prompt (a dollar sign), as follows:

**$** echo "hello, world"

hello, world

Windows users should understand that their systems will use the analogous angle prompt **>**:

C:\Sites> echo "hello, world"

hello, world

On Unix systems, some commands should be executed with **sudo**, which stands for “substitute user do”.[6](http://ruby.railstutorial.org/chapters/beginning#fn-1_6) By default, a command executed with **sudo** is run as an administrative user, which has access to files and directories that normal users can’t touch, such as in this example from[Section 1.2.2](http://ruby.railstutorial.org/chapters/beginning#sec-rubygems):

**$** sudo ruby setup.rb

Most Unix/Linux/OS X systems require **sudo** by default, unless you are using Ruby Version Manager as suggested in [Section 1.2.2.3](http://ruby.railstutorial.org/chapters/beginning#sec-install_ruby); in this case, you would type this instead:

**$** ruby setup.rb

Rails comes with lots of commands that can be run at the command line. For example, in[Section 1.2.5](http://ruby.railstutorial.org/chapters/beginning#sec-rails_server) we’ll run a local development web server as follows:

$ rails server

As with the command-line prompt, the *Rails Tutorial* uses the Unix convention for directory separators (i.e., a forward slash **/**). My Rails Tutorial sample application, for instance, lives in

/Users/mhartl/rails\_projects/sample\_app

On Windows, the analogous directory would be

C:\Sites\sample\_app

The root directory for any given app is known as the *Rails root*, but this terminology is confusing and many people mistakenly believe that the “Rails root” is the root directory for Rails itself. For clarity, the *Rails Tutorial* will refer to the Rails root as the *application root*, and henceforth all directories will be relative to this directory. For example, the **config** directory of my sample application is

/Users/mhartl/rails\_projects/sample\_app/config

The application root directory here is everything before **config**, i.e.,

/Users/mhartl/rails\_projects/sample\_app

For brevity, when referring to the file

/Users/mhartl/rails\_projects/sample\_app/config/routes.rb

I’ll omit the application root and simply write **config/routes.rb**.

The *Rails Tutorial* often shows output from various programs (shell commands, version control status, Ruby programs, etc.). Because of the innumerable small differences between different computer systems, the output you see may not always agree exactly with what is shown in the text, but this is not cause for concern.

Some commands may produce errors depending on your system; rather than attempt the[Sisyphean](http://en.wikipedia.org/wiki/Sisyphus) task of documenting all such errors in this tutorial, I will delegate to the “Google the error message” algorithm, which among other things is good practice for real-life software development. If you run into any problems while following the tutorial, I suggest consulting the resources listed on the [Rails Tutorial help page](http://railstutorial.org/help).[7](http://ruby.railstutorial.org/chapters/beginning#fn-1_7)

[1.2 Up and running](http://ruby.railstutorial.org/chapters/beginning#sec-up_and_running)

*I think of Chapter 1 as the “weeding out phase” in law school—if you can get your dev environment set up, the rest is easy to get through.   
—Bob Cavezza, Rails Tutorial reader*

It’s time now to get going with a Ruby on Rails development environment and our first application. There is quite a bit of overhead here, especially if you don’t have extensive programming experience, so don’t get discouraged if it takes a while to get started. It’s not just you; every developer goes through it (often more than once), but rest assured that the effort will be richly rewarded.

[1.2.1 Development environments](http://ruby.railstutorial.org/chapters/beginning#sec-development_tools)

Considering various idiosyncratic customizations, there are probably as many development environments as there are Rails programmers, but there are at least two broad types: text editor/command line environments, and integrated development environments (IDEs). Let’s consider the latter first.

[IDEs](http://ruby.railstutorial.org/chapters/beginning#sec-1_2_1_1)

The most prominent Rails IDEs are [RadRails](http://www.aptana.com/rails/) and [RubyMine](http://www.jetbrains.com/ruby/index.html). I’ve heard especially good things about RubyMine, and one reader (David Loeffler) has assembled [notes on how to use RubyMine with this tutorial](https://github.com/perfectionist/sample_project/wiki).[8](http://ruby.railstutorial.org/chapters/beginning#fn-1_8) If you’re comfortable using an IDE, I suggest taking a look at the options mentioned to see what fits with the way you work.

[Text editors and command lines](http://ruby.railstutorial.org/chapters/beginning#sec-1_2_1_2)

Instead of using an IDE, I prefer to use a *text editor* to edit text, and a *command line* to issue commands ([Figure 1.1](http://ruby.railstutorial.org/chapters/beginning#fig-editor_shell)). Which combination you use depends on your tastes and your platform.

* **Text editor:** I recommend [Sublime Text 2](http://www.sublimetext.com/2), an outstanding cross-platform text editor that is simultaneously easy to learn and industrial-strength.[9](http://ruby.railstutorial.org/chapters/beginning#fn-1_9) Sublime Text is heavily influenced by [TextMate](http://macromates.com/), and in fact is compatible with most TextMate customizations, such as snippets and color schemes. (TextMate, which is available only on OS X, is still a good choice if you use a Mac.) A second excellent choice is [Vim](http://www.vim.org/),[10](http://ruby.railstutorial.org/chapters/beginning" \l "fn-1_10) versions of which are available for all major platforms. Sublime Text can be obtained commercially, whereas Vim can be obtained at no cost; both are industrial-strength editors, but in my experience Sublime Text is *much* more accessible to beginners.
* **Terminal:** On OS X, I recommend either use [iTerm](http://iterm.sourceforge.net/) or the native Terminal app. On Linux, the default terminal is fine. On Windows, many users prefer to develop Rails applications in a virtual machine running Linux, in which case your command-line options reduce to the previous case. If developing within Windows itself, I recommend using the command prompt that comes with [Rails Installer](http://railsinstaller.org/) ([Section 1.2.2.1](http://ruby.railstutorial.org/chapters/beginning#sec-rails_installer_windows)).

If you decide to use Sublime Text, you might want to follow the optional setup instructions for [Rails Tutorial Sublime Text](https://github.com/mhartl/rails_tutorial_sublime_text).[11](http://ruby.railstutorial.org/chapters/beginning#fn-1_11) (Such configuration settings can be fiddly and error-prone, so I mainly recommend them for more advanced users; Sublime Text is an excellent choice for editing Rails applications even without the advanced setup.)

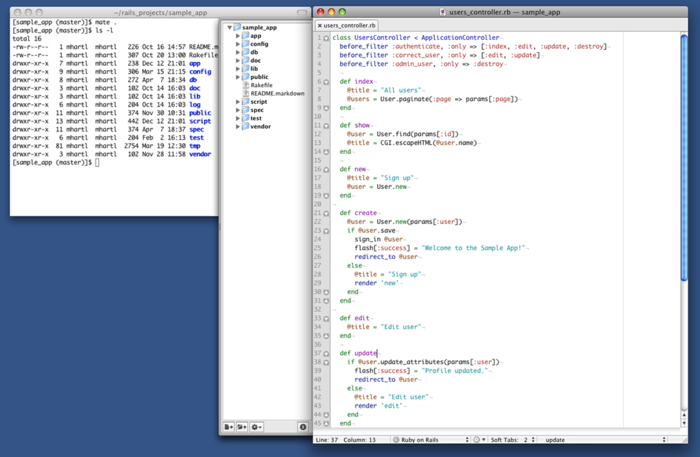


Figure 1.1: A text editor/command line development environment. [(full size)](http://railstutorial.org/images/figures/editor_shell-full.png)

.